MathSTIC

Spécialité : Informatique
Laboratoire : LS2N
Equipe : Naomod, Stack

Evaluation of Combinations of Model Management Execution Strategies for Low-Code Development Platforms

Jolan Philippe
Mél : jolan.philippe@imt-atlantique.fr

Abstract: The low-code abstraction reduces the conception phase of applications. Because of the increasing size of data, abstracting computations raised new challenges. To tackle scalability issues raised by the size of the data, some tools are built upon specific computational strategies exploiting parallelism. However finding the most adapted approach for optimizing the interactivity of the applications is not a trivial problem. Besides, the most efficient solutions may be obtained by the use of several strategies at the same time. This paper motivates the need for a multi-strategy use by executing the same query using different strategies, on different input. It shows a difference of performances, depending on two factors: the input model, and the used strategy.

Keywords: *Spark, Model-Driven Engineering, Social Network*

1 Introduction

Model-Driven Engineering (MDE) has taken an importan place in the development of maintainable software due to its abstraction level. By abstracting concepts, the MDE paradigm reduces the cost and the needed level of expertise both at development and maintenance time [1]. Following this line, low-code engineering provides graphical interfaces for developing applications referred as low-code development platforms (LCDPs) [2]. The conception of applications is based on the manipulation of blocks, which respect a given semantic. They only have an access to these visual models that he can select, combine, or insert, usually following a drag-and-drop approach. At this point, LCDPs users do not have a look on the back-end implementation of these blocks. The fact is the possible implementations can be numerous, with their own benefits and drawbacks. The performance of these operations represent a field of study in the MDE community. More specifically, model-management in LCDPs has a significant need for automatic and transparent efficient and scalable operations, for manipulating, querying and analyzing models.

We identify three main reasons for this need. First, LCDPs need to provide complex visual development environments with low response time for keeping a high level of comfort for developers [3]. Second, there is a need of manipulating large instance models of data (e.g., Facebook graph is about a trillion of relationships [4]). Finally, large number of users may want to manipulate the same data at the same time. LCDPs must then be able to efficiently handle with a huge amount of concurrent operations.

To improve efficiency and scalability, recent research on model-management studied parallel and concurrent programming. These techniques range from implementing specific execution algorithms (e. g., RETE [5]) to compiling toward distributed programming models (e.g., MapReduce [6]). In this paper, we explore the performances of different distributed approaches in a context of querying a social network. Concretely, this paper proposes an extension of previous work published in [7], by providing performance evaluation of parallel queries.

The rest of the paper is organized as follows. We motivate our work with an example in Section 2. Section 3 presents related work from literature that is used for running a such example. In Section **??**, we give a quick overview of our implementations based on distributed approaches. Their evaluation is given in Section 4. We finally give concluding remarks in Section 5

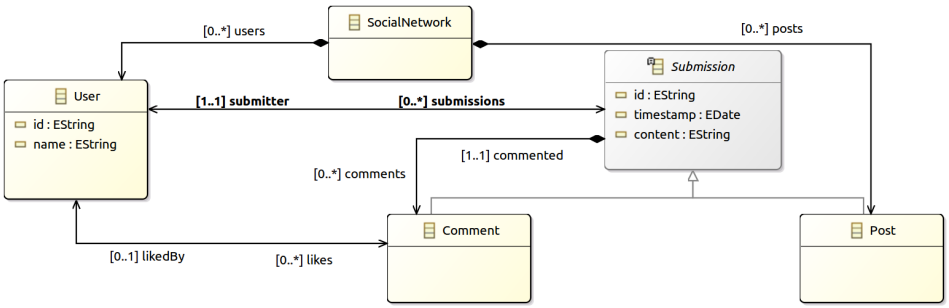

Figure 1: The metamodel of a social network (TTC 2018)

2 Motivating Example

Social network vendors often provide specific development platforms, used by developers to build apps that extend the functionality of the social network. Some networks are associated with marketplaces where developers can publish such apps, and end-users can buy them. Development platforms typically include APIs that allow analyzing and updating the social network graph.

As a running example for this paper, we consider a scenario where a vendor adds a LCDP to allow end-users (also called *citizen developers* in the LCDP jargon) to implement their own apps. Such LCDP could include a 'What you see is what you get editor' for the app user-interface, and a visual workflow for the behavioral part. In particular, the LCDPs would need to provide mechanisms, at the highest possible level of abstraction, to express queries and updates on the social graph.

In Figure 1 we show the simple metamodel for the social graph that we will use in the paper. The metamodel has been originally proposed at the Transformation Tool Contest (TTC) 2018 [8], and used to express benchmarks for model query and transformation tools. In this metamodel, two main entities belong to a `SocialNetwork`. First, the `Posts` and the `Comments` that represent the `Submissions`, and second, the `Users`. Each `Comment` is written by a `User`, and is necessarily attached to a `Submission` (either a `Post` or another `Comment`). Besides commenting, the `Users` can also like `Submissions`.

As an example, in this paper we focus on one particular query, also defined in TTC2018: the extraction of the three most debated posts in the social network. To measure how debated is the post, we associate it with a numeric score. The LCDP will have to provide simple and efficient means to define and compute this score. We suppose the vendor to include a declarative query language for expressing such computation on the social graph, and storing scores as a derived properties of the graph (i.e. new properties of the social graph that are computed on demand from other information in the graph).

3 Related Works

There exist attempts of using distributed strategies for running model management operations. They can be decomposed into three main categories: data-parallelism approach, where the full set of data is split among different processors which applies the same computation on it; task-parallelism where each processor runs a independent computations that not necessary the same; and asynchronism where all data, and tasks are shared between processors.

Data-parallelism This computation strategy has been used by Benelallam et al. [9] for distributing model among computational cores to reduce cores to reduce computation time in the ATL model transformation engine. The MapReduce version of ATL makes independent transformations of subparts of the model by using a local "match-apply" function. They highlight the good impact of their strategy for data partitioning. Instead of randomly distributing the same number of elements among the processors, they use a strategy based on the connectivity of models. at resolving dependencies between map outputs. Graph-based approaches have been proposed. With this approach, the data is structured as a set of vertices, connected by edges. For instance [10] uses Pregel, a framework based on this approach, for computing models with a distributed strategy.

Task-parallelism In [11], Madani et al. use multi-threading for "select-based" operations in EOL, the OCL-like language of the Epsilon framework, for querying models. In [12], Tisi et al. present a prototype of an automatic parallelization for the ATL transformation engine, based on task-parallelism. To do so, they just use a different thread for each transformation rule application, and each match, without taking into account concurrency concerns (e.g., race conditions).

Asyncrhonism LinTra [13] is a Linda-based platform for model management and has several types of implementation. First, on a shared-memory architecture (i.e., a same shared memory between processors, typically multi-threading solutions), LinTra proposes parallel transformations. Nonetheless, shared-memory architecture are fine for not too big models. Indeed, since the memory is not distributed, a too big model could lead to a out-of-memory errors. This phenomenon happens more concretely in an out-place transformation since two models are involved during the operation. The first prototype of distributed out-place transformations in LinTra, is presented in [14].

4 Evaluation

This Section gives an overview of the different strategies we use to implement the query to solve the problem presented in Section 2. All the technical details can be found in [7]. First, the naive implementation is based on its OCL specification. OCL is a standard for expressing queries, and constraints, on elements of models. It is decomposed on several parts: first, all belonging comments of a post are recursively obtained, then all their like is count. From these two information, the score of a post is calculated. This function has a direct, but inefficient, counterpart. The second implementation, designed on top of Pregel, is based on graph theory. Here, each element of the model is considered as a single element. The third implementation works on the MapReduce paradigm: every element of the model, without considering its type, is associated to a score. To finally obtain a score for a port, these intermediate scores a merged into a single value. Finally, we implemented two additional approaches by mixing the one presented above. Some parts of the naive implementation can be optimized using the Pregel strategy. Similarly, the MapReduce implementation can be partially replaced by an execution based on the graph theory of the Pregel paradigm. We experiment our five parallel implementation of the TTC18 query (Section 2). The experiments have been conducted on a shared memory machine with a Intel Core i7-8650U having 8 cores at 1.90GHz and a memory of 32GB. The machine was running Ubuntu 16.04 LTS. We use Java 8, Scala 2.12 with Spark 3.1.0. Each speed-up on Table 1 is the mean of a series of 30 measures.

	Dataset				Speed-up (compared to Naive (Sequential) solution)					
	#users	#posts	#comments	#likes	Naive (Sequential)	Naive (Parallel)	Pregel	MapReduce	OCL + Pregel	MapReduce + Pregel
1	80	554	640	6	**1x**	0.40x	10.30x	5.82x	9.40x	4.63x
2	889	1064	118	24	**1x**	0.39x	0.36x	0.46x	0.44x	0.46x
3	1845	2315	190	66	**1x**	0.51x	0.68x	0.85x	0.66x	0.71x
4	2270	5056	204	129	**1x**	0.27x	0.35x	2.34x	0.15x	2.96x
5	5518	9220	394	572	**1x**	4.25x	5.21x	4.17x	4.68x	4.03x
6	10929	18872	595	1598	**1x**	4.68x	2.83x	2.39x	1.97x	3.91x
7	18083	39212	781	4770	**1x**	4.07x	4.12x	4.58x	5.17x	3.27x
8	37228	76735	1158	13374	**1x**	7.28x	9.52x	7.61x	9.66x	9.22x

Table 1: Speed-up of 5 implementations based on different strategies for the TTC18 query

5 Conclusion

From our experiment, we observe two main points. First, using a distributed strategy generally look too expensive for too small data set (4 first datasets). The second observation is, the input data and its topology has an impact on the performances. For instance, the `Pregel` strategy looks to have a good impact on performances for the 8th dataset, where it has a bad impact for the 6th dataset. The fact that the experiments have been conducted on a single machine can be discussed. Since Spark, the used platform is designed for Cloud architectures, we have to conduct similar experiments on a such environment.

References

[1] Juha Kärnä, Juha-Pekka Tolvanen, and Steven Kelly. Evaluating the use of domain-specific modeling in practice. In *Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling*, 2009.

[2] Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, Juan De Lara, Esther M Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. In *STAF 2019, Software Technologies: Applications and Foundations (STAF 2019)*, CEUR Workshop Proceedings (CEUR-WS.org), Eindhoven, Netherlands, July 2019.

[3] Salvador Martínez Perez, Massimo Tisi, and Rémi Douence. Reactive model transformation with ATL. *Sci. Comput. Program.*, 136:1–16, 2017.

[4] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow.*, 8(12):1804–1815, Aug 2015.

[5] Charles Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligences*, 19(1):17–37, 1982.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, OSDI'04, pages 137–149, Berkeley, CA, USA, 2004. USENIX Association.

[7] Jolan Philippe, Héléne Coullon, Massimo Tisi, and Gerson Sunyé. Towards transparent combination of model management execution strategies for low-code development platforms. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Antonio García-Domínguez, Georg Hinkel, and Filip Krikava, editors. *Proceedings of the 11th Transformation Tool Contest, co-located with the 2018 Software Technologies: Applications and Foundations, TTC@STAF 2018, Toulouse, France, June 29, 2018*, volume 2310 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[9] Amine Benelallam, Abel Gómez, Massimo Massimo Tisi, and Jordi Cabot. Distributed model-to-model transformation with ATL on MapReduce. In Richard R. Paige, Davide Di Ruscio, and Markus Völter, editors, *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering, SLE 2015, Pittsburgh, PA, USA, October 25-27, 2015*, SLE 2015, pages 37–48. ACM, 2015.

[10] Christian Krause, Matthias Tichy, and Holger Giese. Implementing graph transformations in the bulk synchronous parallel model. In Stefania Gnesi and Arend Rensink, editors, *Fundamental Approaches to Software Engineering*, pages 325–339, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[11] Sina Madani, Dimitris S. Kolovos, and Richard F. Paige. Towards optimisation of model queries: A parallel execution approach. *Journal of Object Technology*, 18(2):3:1–21, July 2019. The 15th European Conference on Modelling Foundations and Applications.

[12] Massimo Tisi, Martínez Salvador Perez, and Hassene Choura. Parallel execution of ATL transformation rules. In *Model-Driven Engineering Languages and Systems - 16th International Conference, MODELS 2013, Miami, FL, USA, September 29 - October 4, 2013. Proceedings*, volume 8107 of *Lecture Notes in Computer Science*, pages 656–672. Springer, 2013.

[13] Loli Burgueño, Javier Troya, Manuel Wimmer, and Antonio Vallecillo. Parallel in-place model transformations with LinTra. In *Proceedings on the Software Technologies: Applications and Foundations (STAF 2015) federation of conferences, L'Aquila, Italy, July 23, 2015.*, volume 1406 of *CEUR Workshop Proceedings*, pages 52–62. CEUR-WS.org, 2015.

[14] Loli Burgueño, Manuel Wimmer, and Antonio Vallecillo. A Linda-based platform for the parallel execution of out-place model transformations. *Information & Software Technology*, 79(C):17–35, Nov 2016.